Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Artificial Intelligence & Machine Learning

Object Detection System

Project-II

BACHELOR OF TECHNOLOGY
(Artificial Intelligence and Machine Learning.)

SUBMITTED BY:

Aditya Vishwakarma 2129939
Akash Kumar 2129940
Akshil Thakur 2129942
Alok Ranjan 2129944

August 2024

Under the Guidance of

Ms. Meenakshi

Assistant Professor(CSE)

Department of Artificial Intelligence & Machine Learning

Chandigarh Engineering College Jhanjeri Mohali - 140307

Table of Contents

Introduction

The **Object Detection System** project aims to develop a robust and efficient model for identifying and localizing objects within images and video streams. The system utilizes advanced machine learning techniques, particularly Convolutional Neural Networks (CNNs), to achieve high accuracy and real-time performance. With applications across diverse fields such as security, healthcare, retail, and autonomous driving, the system is designed to meet the growing demand for intelligent automation. The primary objectives include creating a high-accuracy detection model, achieving real-time processing, ensuring adaptability to various environments, and minimizing computational overhead for efficient deployment on multiple platforms.

Key features of an image detection system include real-time processing capabilities, adaptability to different environments and conditions, scalability to handle large volumes of data, and the ability to continuously learn and improve from new data inputs. With the rise of big data and the proliferation of high-resolution imaging devices, the demand for robust and efficient image detection systems is growing rapidly, making them a critical component in modern technological infrastructure.

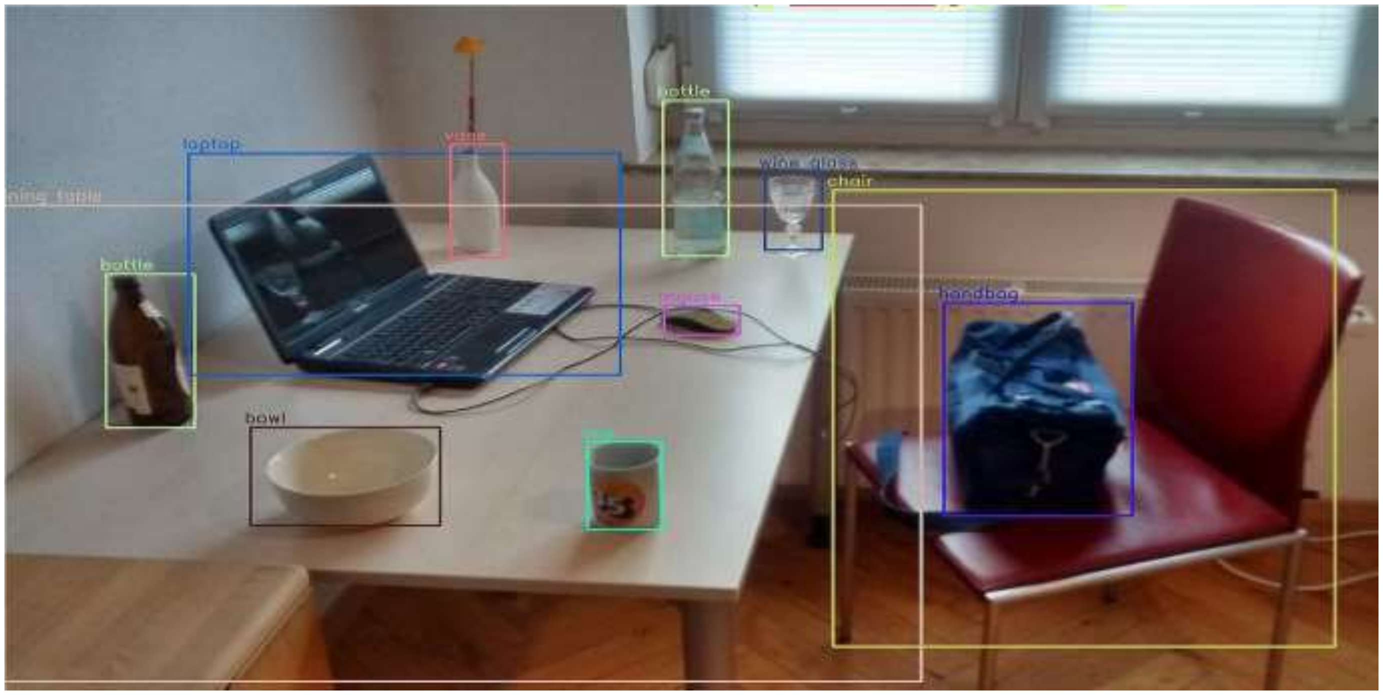

FIG: 1.1

System Requirements

1. Hardware Requirements

Basic Development

- **Processor**: Intel i5 (8th Gen or above) / AMD Ryzen 5 or equivalent
- **RAM**: 8 GB (16 GB recommended for larger datasets)
- **Storage**: 256 GB SSD (minimum), additional HDD for data storage
- **Graphics Card (GPU)**: NVIDIA GTX 1050 Ti or higher with CUDA support
- **Display**: Full HD (1920x1080) monitor

High-Performance Development

- **Processor**: Intel i7/i9 or AMD Ryzen 7/9
- **RAM**: 32 GB or more
- **Storage**: 512 GB SSD (for OS and applications) + 1 TB HDD (for datasets)
- **Graphics Card (GPU)**: NVIDIA RTX 3060 or higher (preferably with Tensor cores)
- **Additional Hardware**:
  - GPU cooling system for sustained high performance.
  - High-speed internet connection for downloading pre-trained models and datasets.

Software Requirement Analysis

1. Software Requirements

Operating System

- Windows 10/11 (64-bit)
- Ubuntu 20.04 or higher (recommended for machine learning projects)
- macOS (if working on Apple Silicon)

Development Environment

- **Python Version**: Python 3.7 or higher
- **Integrated Development Environment (IDE)**: PyCharm, VS Code, Jupyter Notebook

Libraries and Frameworks

- **Core Libraries**:
  - NumPy
  - Pandas
  - Matplotlib / Seaborn (for visualization)
- **Machine Learning Frameworks**:
  - TensorFlow (version 2.x)
  - PyTorch (version 1.x or higher)
- **Object Detection Frameworks**:
  - OpenCV
  - YOLO (You Only Look Once)
  - Faster R-CNN (using PyTorch/TensorFlow)
- **Other Tools**:
  - scikit-learn
  - Pillow (for image processing)
  - Albumentations (for data augmentation)
  - LabelImg or Roboflow (for dataset annotation)

Version Control

- Git (GitHub, GitLab, or Bitbucket for repository management)

Optional Tools

- Docker (for containerization and consistent environment)

- Anaconda (for managing dependencies)

3. Dataset Requirements

- Dataset storage capacity: Large-scale datasets (e.g., COCO, Pascal VOC) require additional storage (at least 500 GB free space recommended).
- Pre-labeled datasets or a labeling tool.

4. Additional Considerations

- **For Real-Time Applications**:
    - Consider edge devices like NVIDIA Jetson Nano or Google Coral for deployment.
    - Ensure low-latency GPU processing.
- **Cloud Resources (Optional)**:
    - Google Colab or Kaggle Notebooks (for free GPU access).
    - AWS, Azure, or GCP for scalable resource.

Software Design

1. Design Goals

- **Accuracy**: The system should accurately detect objects in various environments.
- **Efficiency**: Optimize performance for real-time or batch processing.
- **Scalability**: Handle increasing data and complexity.
- **Usability**: Provide an intuitive interface for end-users or developers.

2. Architecture Overview

A. Layers of the System

1. **Input Layer**:
   - Accepts image or video input from cameras, file uploads, or streaming sources.
2. **Preprocessing Layer**:
   - Normalizes images (resizing, color correction, data augmentation).
3. **Core Detection Layer**:
   - Detects objects using pre-trained models or custom-trained models.
4. **Postprocessing Layer**:
   - Filters and refines detected objects (non-max suppression, confidence thresholding).
5. **Output Layer**:
   - Displays results visually (bounding boxes, labels) or exports data (e.g., JSON, CSV).
6. **Storage Layer**:
   - Saves logs, processed data, and detection results.

3. Key Components

A. User Interface (UI)

- **Type**: Web application, desktop application, or command-line interface.
- **Features**:
  - Upload images/videos.
  - View detection results.
  - Customize parameters (e.g., confidence threshold).
  - Download detection reports.

B. Backend Services

- **Preprocessing Module**:
  - Functions for resizing, scaling, and augmenting images.
- **Inference Module**:
  - Integrates detection models like YOLO, Faster R-CNN, or SSD.
- **Postprocessing Module**:
  - Non-max suppression to eliminate overlapping bounding boxes.
- **Storage Module**:
  - Database (e.g., SQLite, MongoDB) for storing results and metadata.

C. Model Integration

- Support multiple models for flexibility (e.g., YOLOv8, SSD, RetinaNet).
- Use APIs like TensorFlow Model Serving or PyTorch Hub.

D. APIs

- REST or GraphQL APIs for interaction between the UI and backend.
- Model inference endpoints for predictions.

E. Monitoring and Logging

- Track system performance and detection accuracy.
- Tools like Prometheus or ELK stack for analytics.

4. Workflow

Step 1: **Input**

- User uploads or streams an image/video.
- Input is validated and sent to preprocessing.

Step 2: **Preprocessing**

- Normalize input (resize, augment).
- Convert data to model-compatible format (e.g., tensors).

Step 3: **Object Detection**

- Input passes through the detection model.
- Model generates bounding boxes, confidence scores, and class labels.

Step 4: **Postprocessing**

- Apply confidence thresholds.
- Use non-max suppression to refine results.

Step 5: **Output**

- Results are displayed as annotated images or dashboards.
- Data exported in desired format (e.g., JSON, CSV).

Implementation

1. Input Acquisition

The first step in the object detection process is acquiring the input data, which could be an image or video stream. The system should be able to handle various input formats, such as images from cameras, file uploads, or real-time video feeds.

2. Preprocessing

Preprocessing transforms the input data into a suitable format for detection. This step typically includes:

- **Resizing**: Adjusting the input image to the required size for the model.
- **Normalization**: Scaling pixel values to a specific range (e.g., 0-1).
- **Data Augmentation**: Enhancing the dataset with random transformations like rotations, flips, or color adjustments, improving the model's robustness.

3. Object Detection

In this step, the system uses a trained model (such as YOLO, Faster R-CNN, or SSD) to detect objects. These models take the preprocessed image and output:

- **Bounding Boxes**:Coordinates that define the location of detected objects.
- **Class Labels**: Categories to which the detected objects belong.
- **Confidence Scores**: Probability scores indicating how likely the object belongs to a specific class.

4. Postprocessing

After detection, postprocessing refines the results:

- **Non-Maximum Suppression (NMS)**: A technique used to eliminate overlapping bounding boxes by keeping only the one with the highest confidence score.
- **Thresholding**: Filters out low-confidence detections to improve the overall accuracy and reduce false positives.

5. Output

The final step is outputting the results. This could include:

- Annotated images with bounding boxes drawn around detected objects.

- Exporting results in formats like JSON, CSV, or a database, which includes object class, bounding box coordinates, and confidence scores.

6. Evaluation

To measure the performance of the object detection model, several evaluation metrics are used:

- **Precision**: The proportion of true positive detections to all positive detections.
- **Recall**: The proportion of true positive detections to all actual objects in the image.
- **Intersection over Union (IoU)**: A measure of overlap between predicted bounding boxes and ground truth boxes.
- **mAP (mean Average Precision)**: A summary of precision-recall trade-offs across different thresholds.

7. Challenges

Despite its advancements, object detection faces several challenges:

- **Occlusion**: Objects that are partially hidden by others.
- **Scale Variability**: Detecting objects at different sizes.
- **Real-Time Performance**: Balancing speed and accuracy for real-time applications, especially in embedded systems or mobile devices.
- **Class Imbalance**: When some object classes appear more frequently than others, which can bias the model.

REFERENCES

 Research Papers:

- Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "YOLO: You Only Look Once - Unified, Real-Time Object Detection," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. Available: IEEE Xplore.
- Shaoqing Ren, Kaiming He, Ross B. Girshick, Jian Sun, "Faster R-CNN: Towards RealTime Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015. Available: IEEE Xplore.
- Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, ChengYang Fu, Alexander C. Berg, "SSD: Single Shot MultiBox Detector," European Conference on Computer Vision (ECCV), 2016. Available: Springer Link.

 Datasets:

- "COCO: Common Objects in Context," Microsoft. Available: COCO Dataset.
- "PASCAL VOC 2012," Visual Object Classes Challenge. Available: PASCAL VOC Dataset.

 Video Tutorials:

 "Object Detection using YOLO and OpenCV," Tech With Tim. Available: YouTube.
"Faster R-CNN Explained," DeepLizard. Available: YouTube.

 Python Libraries:

- "Numpy Documentation," Available: Numpy.
- "Pandas Documentation," Available: Pandas.
- "TensorFlow Documentation," Available: TensorFlow. "Keras Documentation," Available: Keras.